МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5

по дисциплине «Алгоритмы и структуры данных»

Тема: Случайное бинарное дерево поиска.

Студентка гр. 8304 _____ Мельникова О.А.

Преподаватель _____ Фирсов М.А.

Санкт-Петербург

2019

Цель работы.

Ознакомиться с основными понятиями и реализовать случайное бинарное дерево поиска.

Задание.

Вариант 8

1) По заданному файлу, все элементы которого различны, построить структуру данных типа – случайное БДП.

2) Проверить входит ли в структуру данных элемент E, и если входит, то удалить его. Предусмотреть возможность повторного выполнения с другим элементом.

Описание алгоритма.

Создание БДП было реализовано с помощью метода makeFromFile, который из входного потока считывает элементы и добавляет их в БДП методом SearchAndInsert.

Добавление элемента происходит в случае его неудачного поиска в узлах дерева. Если дерево изначально пустое, то сразу заполняются поля первого узла. Вставка зависит от значения элемента, если он меньше текущего корня, то продолжается поиск в левом поддереве, если больше, то в правом, если же значение value текущего узла равно искомому элементу, то в этом узле увеличивается count. Структура случайного БДП полностью зависит от того («случайного») порядка, в котором элементы расположены во входной последовательности (во входном файле).

Поиск и удаление элемента аналогичен поиску при удалении. В зависимости от значения элемента происходит поиск либо в левом, либо в правом поддереве. При нахождении элемента происходит его удаление. Проще всего его удалить, если этот элемент находится в листе дерева. Тогда данный лист непосредственно удаляется. Если же удаляемый элемент находится во внутреннем узле b, то в ситуации

когда существует правое поддерево, алгоритм находит минимальный элемент правого поддерева, рекурсивно удаляет его и заменяет им содержимое узла b. Если правого поддерева нет, то находится максимальный элемент левого поддерева, рекурсивно удаляется и содержимое узла b заменяется им. При этом если у найденного минимума или максимума есть поддерево (левое, если максимум и правое, если минимум), то оно не удаляется, а первый его узел встает на место удаленного элемента. Методы, реализующие данный алгоритм: SearchAndDelete, GetAndDeleteMaxValue, GetAndDeleteMinValue.

Вывод дерева происходит в обходе ЛКП, при таком обходе элементы выводятся отсортированными в порядке возрастания.

Описание класса.

1. `T value;`

Хранит значение.

2. `unsigned int count;`

Хранит количество повторяющихся элементов.

3. `std::unique_ptr<BinaryTree<T>> leftLeaf;`

Указатель на левое поддерево.

4. `std::unique_ptr<BinaryTree<T>> rightLeaf;`

Указатель на правое поддерево.

5. `void makeFromFile(std::ifstream& fin, std::unique_ptr<BinaryTree<T>>& tree)`

Метод предназначена для создания дерева из входной последовательности элементов.

6. `void SearchAndInsert(T& info, std::unique_ptr<BinaryTree<T>>& tree)`

Метод предназначен для поиска и вставки элемента.

7. `void WriteToFile(std::ofstream& fout, std::unique_ptr<BinaryTree<T>>& tree)`

Вывод узлов в порядке ЛКП.

8. `T GetAndDeleteMinValue(std::unique_ptr<BinaryTree<T>>& tree, unsigned int & count)`

Метод предназначен для нахождения, возврата и удаления минимального элемента.

9. `T GetAndDeleteMaxValue(std::unique_ptr<BinaryTree<T>>& tree, unsigned int & count)`

Метод предназначен для нахождения, возврата и удаления максимального элемента.

10. `int SearchAndDelete(T& info, std::unique_ptr<BinaryTree<T>>& tree)`

Метод предназначен для поиска и удаления элемента.

Тестирование.

Входные данные	Выходные данные
Содержимое файла input.txt: 1.324 2 3 45 546 54 54 116 546 546 8 84 564 15 51 53 1898 123 Содержимое файла inputToDelete.txt: 1.324 2 3 45 546 54 54 116 546	Считанное дерево: 1.324 2 3 8 15 45 51 53 54 84 116 123 546 564 1898 Дерево после удаления заданных элементов: 8 15 51 84 564

546 53 1898 123	
Содержимое файла input.txt: 1.324 1.654 2 3 45.7 546.9 54 54 116 546 546 8 84 564 15 51 53 1898 123 5 6 90 234 456 68 123 678 908 345 76 23 807 23 36 Содержимое файла inputToDelete.txt: 90 234 456 68 123 678 908	Считанное дерево: 1.324 1.654 2 3 5 6 8 15 23 36 45.7 51 53 54 68 76 84 90 116 123 234 345 456 546 546.9 564 678 807 908 1898 Дерево после удаления заданных элементов: 1.324 1.654 2 3 5 6 8 15 23 36

345	45.7 51 53 54 76 84 116 546 546.9 564 807 1898
Содержимое файла input.txt: 1.324 Содержимое файла inputToDelete.txt: 1.324 84 564	Считанное дерево: 1.324 Дерево после удаления заданных элементов: 0
Содержимое файла input.txt: Содержимое файла inputToDelete.txt: 1.324 2	Считанное дерево: 0 Дерево после удаления заданных элементов: 0
Содержимое файла input.txt: 116 546 546 8 Содержимое файла inputToDelete.txt: 84 564cd; 15 51 53 1898 123	Считанное дерево: 8 116 546 Дерево после удаления заданных элементов: 8 116 546
Содержимое файла input.txt: asdg csjdhfg zsk Содержимое файла inputToDelete.txt: 5 saldfgh	Считанное дерево: 0 Дерево после удаления заданных элементов: 0

Вывод.

В результате работы был получен опыт по реализации случайных бинарных деревьев поиска, и создана программа на языке си++, удовлетворяющая требованиям.

Приложение А.

Файл main.cpp

```
#include "Tree.h"

int main(int argc, char* argv[]) {
   if(argc <= 2)
   {
      std::cout<<"Не введены аргументы командной строки - название файла для считывания дерева и файла с содержимым, которое нужно удалить из дерева!\n";
      exit(1);
   }
   else
   {
      std::unique_ptr<BinaryTree<double>> tree(new BinaryTree<double>());
      std::ifstream inputFile(argv[1]);
      std::ifstream inputToDeleteFile(argv[2]);
      if (!inputFile.is_open())
      {
         std::cout << "ERROR: file isn't open" << std::endl;
         return 0;
      }
      if (!inputToDeleteFile.is_open())
      {
         std::cout << "ERROR: file isn't open" << std::endl;
         return 0;
      }
      if (inputFile.eof())
      {
         std::cout << "ERROR: file is empty" << std::endl;
         return 0;
      }
      tree->makeFromFile(inputFile, tree);
      inputFile.close();

      std::ofstream outputFile1("ReadTree.txt");
      tree->WriteToFile(outputFile1, tree);
      outputFile1.close();

      double elem;
      int s =0;
      while (inputToDeleteFile >> elem) tree->SearchAndDelete(s, elem, tree);

      std::ofstream outputFile2("TreeAfterDelete.txt");
      tree->WriteToFile(outputFile2, tree);
      outputFile2.close();

      return 0;
   }
}
```

Приложение Б.

Файл Tree.h

```
#pragma once
#include <iostream>
#include <string>
#include <memory>
#include <cstdlib>
#include <cstdio>
#include <fstream>
// если элем удалить
template <typename T>
class BinaryTree
{
public:
    T value;
    unsigned int count = 0;
    std::unique_ptr<BinaryTree<T>> leftTree;
    std::unique_ptr<BinaryTree<T>> rightTree;

    void makeFromFile(std::ifstream& fin, std::unique_ptr<BinaryTree<T>>& tree){
        T info;
        while(fin >> info){
            SearchAndInsert(info, tree);
        }
    }
    void SearchAndInsert(T& info, std::unique_ptr<BinaryTree<T>>& tree){
        if((tree->count) == 0 ) {
            tree->value = info;
            tree->count = 1;
        }else if ( info < tree->value ) {
            if (!tree->leftTree) tree->leftTree = std::make_unique<BinaryTree<T>>();
            SearchAndInsert(info, tree->leftTree);
        }else if(info>(tree->value)){
            if (!tree->rightTree) tree->rightTree = std::make_unique<BinaryTree<T>>();
            SearchAndInsert(info, tree->rightTree);
        }else (tree->count)++;
    }

    void WriteToFile(std::ofstream& fout, std::unique_ptr<BinaryTree<T>>& tree) {
        if (tree->leftTree) WriteToFile(fout, tree->leftTree);
        fout << tree->value << std::endl;
        if (tree->rightTree) WriteToFile(fout, tree->rightTree);
    }

    T GetAndDeleteMinValue(std::unique_ptr<BinaryTree<T>>& tree, unsigned int & count) {
        if (tree->leftTree) {
            return GetAndDeleteMinValue(tree->leftTree, count);
        }
        else {
            T result = tree->value;
            count = tree->count;
            if (tree->rightTree)
                tree->value = GetAndDeleteMinValue(tree->rightTree, tree->count);
            else
                tree = NULL;
```

```
        return result;
    }
  }

  T GetAndDeleteMaxValue(std::unique_ptr<BinaryTree<T>>& tree, unsigned int & count) {
    if (tree->rightTree) {
      return GetAndDeleteMaxValue(tree->rightTree, count);
    }
    else {
      T result = tree->value;
      count = tree->count;
      if (tree->leftTree)
        tree->value = GetAndDeleteMaxValue(tree->leftTree, tree->count);
      else
        tree = NULL;
      return result;
    }
  }


  int SearchAndDelete(int& count, T& info, std::unique_ptr<BinaryTree<T>>& tree){
    if (tree->value != info) {
      if(info < tree->value) {
        if(tree->leftTree != NULL) {
          count++;
          return SearchAndDelete(count, info, tree->leftTree);
        }
        else return -1;
      } else {
        if(tree->rightTree != NULL) {
          count++;
          return SearchAndDelete(count, info, tree->rightTree);
        }
        else return -1;
      }
    }else{
      if (tree->rightTree) {
        tree->value = GetAndDeleteMinValue(tree->rightTree, tree->count);
      }
      else if (tree->leftTree) {
        tree->value = GetAndDeleteMaxValue(tree->leftTree, tree->count);
      }
      else {
          if(count == 0){
                tree->value = 0;
                tree->count = 0;
          }else{
          tree = NULL;
        }
      }
      return 1;
    }
  }
};
```